基于公理设计的软件构件设计优化

吴克寿[1,2*],刘丽娟[2]
(1. 华中科技大学机械科学与工程学院,湖北 武汉 430074;2. 厦门理工学院计算机科学与技术系,福建 厦门 361024)

摘要:为了提高软件开发的有效性和正确性,提出了一种基于公理设计的构件设计与优化新方法,详细给出了该方法的设计和实现过程.在此基础上,以调度构件的开发作为研究实例,分别对按照公理设计和传统方法得到的两种设计方案进行了设计、分析、比较和优化,验证了该方法在软件开发方面的有效性和正确性.

关键词:公理设计;构件;软件设计;优化
中图分类号:TP 311 **文献标志码**:A **文章编号**:0438-0479(2012)01-0041-05

公理设计(axiomatic design,AD)理论是美国麻省理工学院(MIT)Suh 教授于 1990 年在《The Principle of Design》一书中正式提出的.AD 理论是设计领域内的科学准则,能指导设计者在设计过程中做出正确的决策,为创新设计或改善已有的设计提供良好的思维方法[1].随着对 AD 研究的深入,AD 理论正在逐步被应用于许多领域,其中也包括软件设计领域.与此同时,基于构件的软件设计技术在软件业被广泛推广和使用,它的灵活性、重用性、可维护性和高效性为现代化软件生产工程化的发展提供了很好的技术支持[2,3].

基于 AD 理论和构件的软件开发技术,本文对基于 AD 的构件设计与优化进行了研究.公理化设计作为产品科学的设计方法之一,在欧美日等国家受到高度重视并得到了极大的发展.2000 年 6 月在英国剑桥召开了第一届公理化设计国际研讨会[4].国内外许多学者在该领域也做了较多的研究[5-9].尽管很多研究给出了相关实例,但是直接用于软件设计领域的实例并不多,且并没有针对独立性和信息量最小性对设计实例进行设计、分析和优化.因此,本文从 AD 理论和基于构件的软件开发理论出发,提出了一种基于 AD 理论的软件构件设计优化方法.通过对构件的设计性能进行分析和验证,指出了其存在的问题.在此基础上,以 AD 为基础的构件开发为指导思想,提出了最优的设计方案,重新对构件进行了设计,验证了 AD 理论在软件设计过程中的可行性和有效性.

收稿日期:2011-03-01
基金项目:福建省自然科学基金项目(2008J0318)
*通信作者:kswu@xmut.edu.cn

1 基于 AD 的软件构件设计方案

在公理设计中,将软件设计分成 4 个区域:用户域、功能域、物理域和过程域,如图 1 所示.相邻域之间的关系是"我们想实现什么"和"如何实现我们想实现的".域和域之间通过映射矩阵建立关系,层层映射进行"之字形"分解,从而将设计任务分配到不同的设计域中完成.

图 1 AD 理论中软件设计的 4 个区域
Fig. 1 Four domains of software design based on AD

图 2 给出了基于 AD 的软件构件设计方案,具体过程如下:

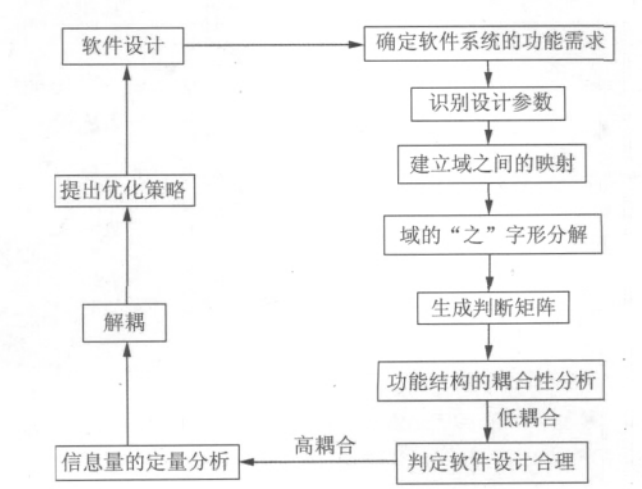

图 2 基于 AD 的软件构件设计方案
Fig. 2 The processing of software design based on AD

1) 确定软件系统的功能需求

根据用户域的信息,确定软件系统必须满足的用户需求和属性,然后确定基本的功能要求(function requires,FRs)和约束(Constraints,Cs),保证用户需求的完备性. 为了满足 AD 的独立公理,功能要求的个数应尽可能少且相互独立.

2) 识别设计参数

进行功能域和结构域之间的"之字形"分解和映射变换,制定合适的设计参数(design parameters,DPs),以满足特定的 FRs 和 Cs,直到功能要求和设计参数分解到不需要分解为止. 此时,将得到一个功能要求和设计参数的等级结构,其中"叶"级设计参数是软件的最终组成部分,即对应的颗粒度适合的构件.

3) 生成判断矩阵 $\boldsymbol{A}$

根据"之字形"分解功能域的 FRs 和结构域的 DPs,得到判断矩阵 $\boldsymbol{A}$. 形式为$\{FR\}=\boldsymbol{A}\{DP\}$.

4) 功能结构的耦合性分析

对判断矩阵 $\boldsymbol{A}$ 进行耦合性分析. 如果 $\boldsymbol{A}$ 为对角矩阵,则设计为线性设计,A_{ij} 是常数;而对于非线性设计,A_{ij} 是 DPs 的函数. 满足独立性公理的设计矩阵 $\boldsymbol{A}$ 有两种典型形式:对角阵和上/下三角阵. 当 $\boldsymbol{A}$ 为对角阵时,设计称为非耦合设计,非耦合设计是理想的设计;当 $\boldsymbol{A}$ 为上/下三角阵时,设计称为准耦合设计. 若 $\boldsymbol{A}$ 为其他一般形式,则设计是耦合设计. 非耦合设计和准耦合设计满足功能独立性公理,而耦合设计不满足功能独立性公理,必须进行解耦处理或重新设计.

5) 最佳设计

通过信息量定量分析,修改"叶"级的 DPs,即构件的实现参数,使得重新生成的 $\boldsymbol{A}$ 是对角阵或上/下三角阵的形式,从而提出解耦的优化策略. 此策略满足信息公理的最小信息量设计标准,即为最佳设计.

基于 AD 的软件构件的设计方案,从软件的功能入手,在设计初期应用 AD 进行分析,得到软件等级结构、流程图和构件间的关联和操作序列. 这种设计方法充分结合了 AD 和软件构件技术的优点,从而得到比较理想的设计结果.

2 实例分析

基于构件的虚拟实验室是本文的应用实例. 我们将计算机领域的一些课程中的实验仪器、调度过程、连接过程均进行构件的封装,然后按照实验流程选择合适的构件来组装实验,从而实现计算机类课程虚拟实验室中实验的设计和分析仿真过程[10]. 在一次实验流程中,构件被分为功能构件(functional component,FC)、连接构件(connecting component,CC)和调度构件(scheduling component,SC). FC 分为数据源产生构件(source functional component,SFC)、处理功能构件(processing functional component,PFC)和显示功能构件(displaying functional component,DFC). CC 依赖于 FC 上的输入引脚和输出引脚分别产生输入连线(input connector,IC)和输出连线(output connector,OC). CC 根据连接在 FC 上引脚的不同分为 IC 和 OC 两种属性. 上一级 FC 的 OC 从其输出引脚相连到下一级 FC 的 IC 的输入引脚上,即上一级 FC 的输出引脚就是下一级 FC 的输入引脚. 其中 SFC 没有输入引脚,DFC 没有输出引脚. 构件间的调度过程封装在 SC 中实现. 图 3 给出了在一次实验流程中构件间的数据流通信模型.

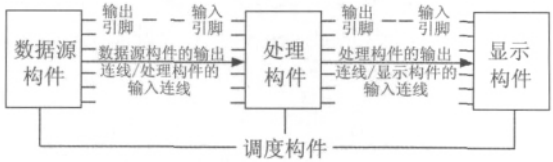

图 3 构件间的数据流通信模型

Fig. 3 The communication module of data flow among components

2.1 基于 A_1 的调度构件的设计

构件的调度策略受到构件模型、构件粒度、运行环境和工作过程等因素的制约. 通过自定义一些优先级、循环、排队、时间片轮询、抢占等策略,动态地建立起各个相互独立的构件间的因果联系,使它们组合成颗粒度大的构件或者一个能够协调工作的系统,从而最大限度地利用各种资源完成多任务并发和实时的工作过程. 我们将具有这些功能的构件定义为调度构件,其功能需求单独抽取出来,对它进行"之字形"分解,得到了一种可以调度构件间数据流具有单向、非连续特点的 SC 的 FRs 以及对应的 DPs,其设计矩阵 $\boldsymbol{A}_1$ 如表 1 所示. 可以看出,SC 根据构件间数据流的特点,分成了顺序调度、并行调度和循环调度这 3 个子功能. 通过引入被调度构件的拓扑结构和构件间数据流复杂度 2 个 DPs,对调度构件的 FR_1 进行了"之字形"分解,得到的设计矩阵 $\boldsymbol{A}_1$ 如下所示:

$$\boldsymbol{A}_1=\begin{Bmatrix}FR_{11}\\FR_{12}\\FR_{13}\end{Bmatrix}=\begin{Bmatrix}X & 0\\X & X\\X & X\end{Bmatrix}\begin{Bmatrix}DP_{11}\\DP_{12}\end{Bmatrix}.$$

表 1　单向非连续数据流驱动的调度构件的判断矩阵 A_1

Tab. 1　The design matrix A_1 of the unidirectional and discontinuous data flow-driven scheduling component

FR_1 调度构件功能需求	DP_1 设计参数	
	DP_{11} 拓扑结构	DP_{12} 数据流的复杂度
FR_{11} 控制构件间数据流顺序传递的调度过程	X	0
FR_{12} 控制构件间数据流并行传递的调度过程	X	X
FR_{13} 控制构件间数据流循环传递的调度过程	X	X

从 A_1 可以看出,由于 DPs 的数目小于 FRs 的数目,根据 AD 理论的定理 1[2],我们可以判定,A_1 并不是理想的对角阵或上/下三角阵的形式,这个设计是一个耦合设计,不是最优的设计结果.

基于 A_1 的调度构件的算法实现如图 4 所示,具体的工作过程如下:

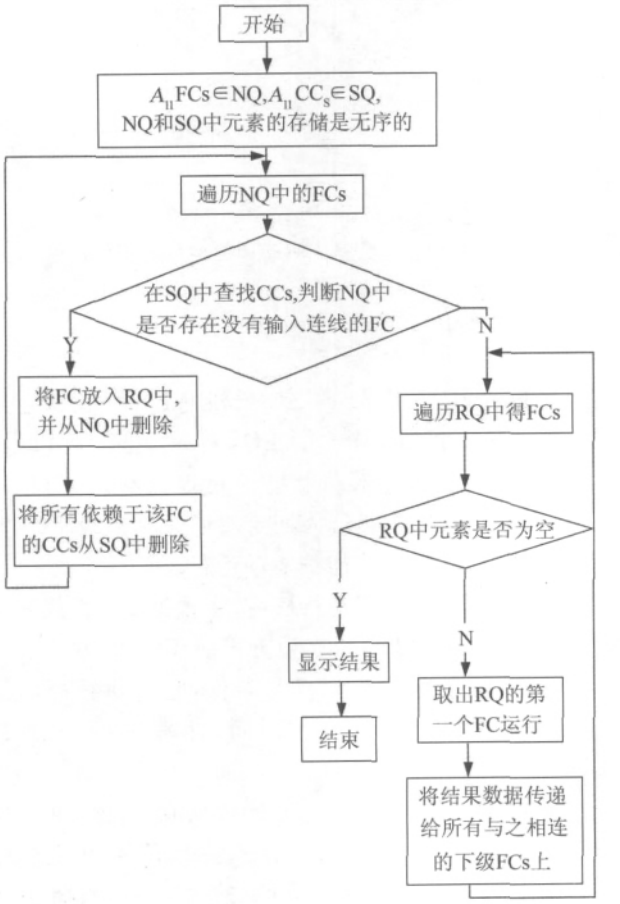

图 4　单向非连续数据流驱动的调度构件的工作过程

Fig. 4　The processing of the unidirectional and discontinuous data flow-driven scheduling component

1) 遍历并初始化一次工作流程中的所有 FC 和 CC,并将它们以无序的方式存储在节点队列(node queue,NQ)和边队列(siding queue,SQ)中. 这时的接口记录了 FC 和 CC 的连接关系,且有 A_{11} FCs∈NQ,A_{11} CCs∈SQ,NQ 和 SQ 中元素的存储顺序是无序的.

2) 遍历 NQ 中的 FCs,直到找到没有 IC 的 FC,这样就找到了 SFC,即工作流程中的产生源数据流的 FC,把它映射到 RQ,并为它创建一个单独的线程处于就绪的状态,同时将它从 NQ 中删除.

3) 在 SQ 中找到与该 FC 相连的所有 CCs,并将它们从 SQ 中删除. 这样,NQ 中的与 SFC 相连的 FCs 的连接属性发生了变化,它们变成了没有输入引脚的 SFC.

4) 重复 2),3)步,使得所有的 FC 按照工作流程顺序映射到 RQ 中.

5) 按照 RQ 的逻辑存储顺序,依次调用 FCs,完成数据的处理和传递过程,接着将它们从 RQ 中删除,直到最后 DFC 运行,从而结束整个工作流程,系统处于稳定的状态.

2.2　基于 A_1 的 SC 分析

这种类型的 SC 可以调度构件间数据流单向、不可逆、分层、顺序的特点,适合完成基于构件的密码学虚拟实验室的调度过程. FCs 存在明显的层次关系,后一级的 FCs 的输入数据依赖于与之相连的上一级的 FCs 运行产生的数据流,但对上一级的 FCs,数据流并没有构成环形结构. 当 FCs 间的数据流关系变得稍微复杂时,如双向、循环、反馈等关系,这种 SC 就显得无能为力了. 在一次实验流程中,拓扑结构是根据 FCs 有无输入连线的属性动态建立和得到的,拓扑结构一旦得到,将不会改变,不适合拓扑结构在运行过程中动态变化的系统,缺乏自适应调整的灵活性. 因此,基于这种单向非连续数据流驱动的 SC 的适用范围具有很大的局限性. 这与基于 AD 理论分析出的设计矩阵 A_1 并不是理想矩阵的判断依据也是一致的. 我们需要重新设计一个满足非耦合设计的新矩阵 A_2,实现最佳设计.

2.3　基于 A_2 的 SC 的设计和优化

鉴于单向非连续数据流驱动 SC 的局限性,需要利用 AD 理论的设计原理,对 A_1 进行修改,以求达到最优的设计效果. 根据 AD 理论的定理 4[2],任何具有相等的 FRs 和 DPs 数目的设计,当设计矩阵是一个对角形或三角形的,总是满足独立公理,在设计中,不会

发生耦合,其设计必然是理想设计. 基于此定理,我们在 SC 中添加了一个 DP——标志位,完成了双向触发式数据流驱动 SC 的新设计矩阵 A_2,如表 2 所示.

表 2 双向触发式数据流驱动 SC 判断矩阵 A_2

Tab. 2 The design matrix A_2 of the bidirectional and triggered dataflow-driven scheduling component

FR_1 调度构件功能需求	DP_1 设计参数		
	DP_{11} 拓扑结构	DP_{12} 数据流的复杂度	DP_{13} 标志位
FR_{11} 控制构件间数据流顺序传递的调度过程	X	0	0
FR_{12} 控制构件间数据流并行传递的调度过程	X	X	0
FR_{13} 控制构件间数据流循环传递的调度过程	X	X	X

由表 2 可以看出,在原来的基础上,通过增加标志位,得到了新的设计矩阵 A_2 如下所示:

$$A_2=\begin{Bmatrix}FR_{11}\\FR_{12}\\FR_{13}\end{Bmatrix}=\begin{Bmatrix}X & 0 & 0\\X & X & X\\X & X & X\end{Bmatrix}\begin{Bmatrix}DP_{11}\\DP_{12}\\DP_{13}\end{Bmatrix}.$$

A_2 使得重新设定的矩阵满足下三角矩阵的形式. 从 A_2 可以看出,该设计对原设计矩阵进行了解耦的操作. 无耦合设计从理论上保证了设计的最优化,使其应用范围更广,功能更全面和高效.

基于此设计过程,我们在已有的单向非连续数据流驱动的 SC 设计的基础上进行了修改,实现了双向触发式数据流驱动 SC,其算法实现如图 5 所示,具体的工作过程如下:

1) 遍历并初始化一次工作流程中的所有 FCs 和 CCs,并把它们随机地存储在 NQ 和 SQ 中. 这时有 A_{11} FCs∈NQ,A_{11} CCs∈SQ,FCs 和 CCs 中元素的存储顺序是无序的.

2) 初始状态,遍历 NQ 中的所有 FCs,依次给它们建立一个线程,即 NQ 中所有的 FCs 都被装载到了同一个线程组中. 同时,根据 FC 上输出引脚记录的连接关系,为每个 FC 建立一张连接关系表(connected form,CF). 该表记录了每个 FC 的输出引脚的数据值和连接关系,即保存了工作流程中的拓扑结构. 对于 FCs 的每个输出引脚,都有一个标志位 Flag 来控制对数据的操作. 这时,所有 FCs 的输出引脚的值都为空,Flag 的值为 0.

3) 启动线程组,任意一个 FC 将以等概率的机会获得运行权. 根据NQ中FCs有无ICs的条件判断得到运行权的FC是否是SFC. 如果没有IC,则表示它是SFC. 除 SFCs 以外,其他的 FCs 都会在运行时调用 wait()方法使其处于等待状态而迅速放弃运行权. 因此,这些线 FCs 真正运行周期需要通过其上级 FCs 调用 notify()方法唤醒才能被驱动运行.

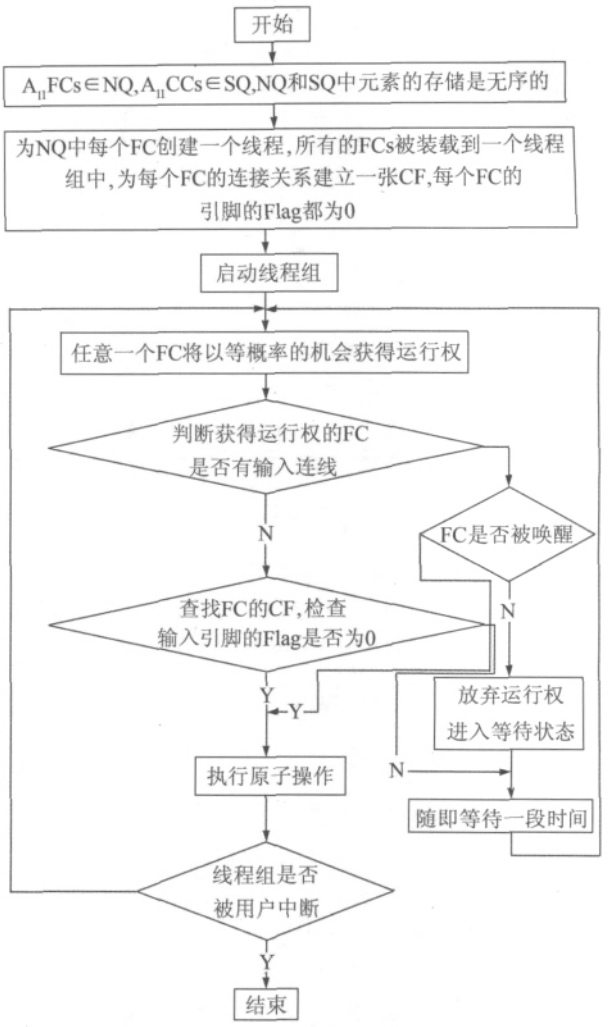

图 5 双向触发式数据流驱动调度构件的工作过程

Fig. 5 The processing of the bidirectional and triggered dataflow-driven scheduling component

4) SFCs 一旦获得运行权后,将根据 CF 查找与它相连的所有下级 FCs,依次检查下级 FCs 的输入引脚的 Flag 是否全为 0,如果满足条件,它将执行原子操作. 否则,它将随机等待一段时间,直到下次运行权的获得.

5) 原子操作包括:产生原始数据流;根据 CF 依次调用 notify()唤醒与它相连的下一级 FCs,并把产生的数据传递到对应关系的 FCs 的输入引脚;将下级 FCs 的输入引脚的 Flag 置为 1,自身的输入引脚的 Flag 置为 0;重新调用 wait()进入等待状态.

6) 线程组中所有的 FCs 一旦被上级 FCs 唤醒,

都将以 4)、5)的方式来被驱动和调度执行,直到系统处于稳定的工作状态,即 FCs 中的数据流不发生任何的变化,最后由 DFCs 显示工作流程的结果.

7) 在 FCs 运行的过程中,系统还将实时检测用户的停止响应,一旦接收到用户停止的消息,整个线程组将被中断,系统工作结束. 图 5 给出了该种 SC 的工作过程.

由此可见,基于 A_2 设计的双向触发式数据流驱动 SC 弥补了基于 A_1 设计的单向非连续数据流驱动的 SC 的局限性. 构件间数据流的依赖性强,且具有良好的可控性和记忆性. 该 SC 不仅有效地保证了数据流连续、循环、反馈等传递过程的实现,又兼有单向非连续数据流驱动的 SC 的特点,使其适用范围更广,是一个理想的非耦合设计. 基于此我们设计和实现了基于构件的数字信号处理[11]和计算机组成原理虚拟实验室[12].

3 结　论

通过对 AD 理论和软件构件设计的深入研究和总结,提出了基于公理设计的软件构件的分析和优化设计的研究思路. 将以构件的软件实现过程从直接按工作流程实现映射到设计一个理想三角设计矩阵的理论层次进行分析. 实例中,通过对比两个不同的 SC,实现了软件设计的解耦和优化过程,从而验证了 AD 理论对软件设计及优化的可行性和正确性.

参考文献:

[1] Suh N P. Axiomatic design: advances and applications [M]. New York: Oxford University Press, 2001.

[2] Sun Changai, Liu Chao, Jin Maozhong, et al. Architecture framework for software test tool [C]//Technology of Object-Oriented Languages and Systems. Xi′an, China: IEEE Computer Society Press, 2000: 40-47.

[3] Quan Bingzhe, Yu Jiang, Jin Chunzhao. Sofware reuse techniques based on component [J]. Journal of Chinese Computer Systems, 1991, 12(9): 38-42.

[4] 郑称德. 公理化设计基本理论及其应用模型[J]. 管理工程学报, 2003, 17(2): 81-84.

[5] Melvin J W, Suh N P. Simulation within the axiomatic design framework[J]. Annalsof the CIRP, 2002, 15(1): 107-110.

[6] 宋慧军, 林志航. 公理化设计支持的概念设计产品模型[J]. 计算机辅助设计与图形学学报, 2002, 14(7): 632-636.

[7] 朱龙英, 朱如鹏. 基于公理化设计理论的并行设计研究[J]. 机械设计, 2003, 20(4): 51-53.

[8] Morc A, Enciso M. An efficient preprocessing transformation for functional dependencies sets based on the paradigm[J]. Lecture Notes in Computer Science, 2004, 30(4): 136-146.

[9] 曹鹏彬, 肖人彬, 库琼. 公理设计过程中耦合设计问题的结构化分析方法[J]. Journal of Mechanical Engineering, 2006, 42(3): 46-55.

[10] Wu Keshou, Liu Lijuan. The design and implementation of image compression comparator in a component-based simulation system for digital image processing [C]// World Congress on Software Engineering, 2009 WRI World Congress on Software Engineering (Volume 3). Xiamen: IEEE Computer Society Press, 2009: 350-353.

[11] Wang Jianxin, Liu Lijuan, Jia Weijia. The design and implementation of digital signal processing virtual lab based on components [C]//The 4th International Conference on Web-based Learning (ICWL 2005). Germany: Springer Press, 2005: 291-301.

[12] Wang Jianxin, Zhang Liyuan, Sheng Yu, et al. Design and Implementation of principles of computer organization virtual lab based on component [J]. Journal of System Simulation, 2008, 20(9): 2469-2474.

Research on Software Design and Optimization Based on Axiomatic Design and Component

WU Ke-shou[1,2*], LIU Li-juan[2]

(1. School of Mechanical Science & Engineering, Huazhong University of Science & Technology, Wuhan 430074, China;
2. Department of Computer Science & Technology, Xiamen University of Technology, Xiamen 361024, China)

Abstract: Aiming at improving the efficiency and validity, a new software architectwre design optimization method is proposed based on axiomatic design and a detailed implementation process is given. This study gives an example of a scheduling component design in two differents methods. By analysing, comparing and optimizing the two methods, it illustrates the effectiveness and reliability in software design through an interactive process between axiomatic design and components.

Key words: axiomatic design; component; software design; optimization